IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

(A) Identification

Appellant	:	Sabin Belu	Confirmation No.	:	8657
Application No.	:	09/818,134	Group Art Unit	:	2166
Filing Date	:	March 27, 2001	Examiner	:	Channavajjala, Srirama T
Docket No.	:	RN058 (2635-019-03)	Customer No.	:	72455
Title	:	SYSTEMS AND METHODS FOR CREATING SELF-EXTRACTING FILES			

APPEAL BRIEF
37 CFR §41.41

Sir:

This is an appeal pursuant to 37 C.F.R. § 1.134(a) from the decision of the Examiner, dated July 6, 2009, finally rejecting claims 1-34 of the above-referenced patent application.

This Appeal Brief is filed within two months from the mailing date of the USPTO's Notice of Panel Decision from Pre-Appeal Brief Review pursuant to 37 C.F.R. § 1.136, along with a one month extension of time and requisite fee.

37 CFR §1.8
CERTIFICATE OF TRANSMISSION

I hereby certify that this correspondence is being transmitted via the Office electronic filing system, EFS-Web, addressed to the Commissioner for Patents, PO Box 1450, Alexandria, Virginia 22313-1450 on the date indicated below.

/Paola Kuvac/ March 12, 2010

Paola Kuvac Date

(B) Table of Contents

(C) Real Party in Interest

The real party in interest is RealNetworks, Inc., the assignee of record, having a principal place of business at 2601 Elliott Ave., Suite 1000, Seattle, Washington 98121-3370.

(D) Related Appeals and Interferences

Based on information obtained from RealNetworks, Inc., and based on information and belief of the undersigned agent, there are no related interferences, appeals, or judicial proceedings known to Appellant, Appellant's agent, or the Assignee, which are related to, directly affect or are directly affected by, or which have a bearing on the decision of the Board in the pending Appeal.

(E) Status of Claims

Claims 1-34 are pending and stand rejected by the Examiner. No claims are allowed. The rejection of claims 1-34 is appealed.

(F) Status of amendments

The Appellant's agent has not amended any of the claims subsequent to the advisory action mailed on September 14, 2009, which indicated entry of all pending amendments. ***Appendix A*** includes all the appealed claims 1-34 as currently pending.

(G) Summary of claimed subject matter

This summary provides cross-referencing to the application as required by 37 C.F.R. § 41.37(c)(v). This cross-referencing is solely to assist the Board in locating portions of the written description and drawings that relate to claimed subject matter and is not meant to be exhaustive or to be used in interpreting the scope of the pending claims.

Claim 1

Claim 1 recites:

> "A method for creating, in response to only a single action by a user enabled electronic device, a self-extracting file, the method comprising:
>
> receiving from the user enabled electronic device, an input file to be used in creating a self-extracting file; and
>
> without further action by the user enabled electronic device, creating a self-extracting file using the input file, wherein the input file is configured to be automatically launched upon execution of the self-extracting file."

The written description and drawings disclose various embodiments related to the subject matter of claim 1.

An embodiment of "receiving from the user enabled electronic device, an input file to be used in creating a self-extracting file" is described at FIG. 4, 402; and P: 16; L: 22-23: "Beginning at a start state 400, the creation process proceeds to the next state 402, wherein an input file 202 is received."

An embodiment of "without further action by the user enabled electronic device" is described at P: 20; L:2-4: "Once the name of the file to be converted into a universal self-extracting file appears in text window 502 [FIG. 5], the user performs a single action, such as clicking a 'Create File' button, to create the universal self-extracting file 204."

An embodiment of "creating a self-extracting file from the input file" is described at FIG. 4 404 – 420, and at P:16; L: 20 – P: 19; L:26.

An embodiment of "wherein the input file is configured to be automatically launched upon execution of the self-extracting file" is described at FIG. 4 412 and at P: 18; L: 4-6: "A loader is attached to the universal self-extracting file 204, which does not yet contain the archive header 208 or compressed input file data 206, in state 412 to automatically launch the input file 202 upon execution of the self-extracting file." An embodiment is further described at FIG. 6 and at P: 21; L: 8-11: "After decompression of the file to its original non-compressed form, the file is launched in state 608 with the appropriate application software for the given file. In on embodiment, for example, this is accomplished by calling the appropriate 'OPEN WITH' application registered with the current operating system."

Claim 2

Claim 2 recites: "The method of claim 1, wherein the received input file has an associated filename and wherein a filename for the self-extracting file is configured to be automatically generated based in part on the associated filename of the received input file."

An embodiment of "wherein the received input file has an associated filename and wherein a filename for the self-extracting file is configured to be automatically generated based in part on the associated filename of the received input file" is described at P: 16; L: 29 – P: 17; L: 3: "In one embodiment, the name is generated based on the name and/or file type of the original input file 202 received by the compression module 108. For example, the name of the universal self-extracting file 204 may be generated based on the following pattern: FILENAME_FILEEXTENSION.EXE. Thus if the input file 202 is named 'house.bmp,' the produced universal self-extracting file 204 would be named 'house_bmp.exe.'" An embodiment is further described at FIG. 5 and P: 22; L: 27-31: "After receving the file, the universal self-extracting systems and methods use the filename of the digital photograph to create a name for an output file [that] is opened (i.e., created) and named 'George's Birthday_Jpeg_Lipon.exe.'"

Claim 3

Claim 3 recites: "A method for creating, in response to a single action, a self-extracting file from an associated input file, wherein the associated input file is automatically launched upon execution of the self-extracting file, and wherein a user is not required to separately choose a data compression method, create a compressed archive using the chosen compression method, select an input file to be launched upon decompression of the compressed archive, and create a self-extracting file from the compressed archive, the method comprising:

receiving an input file to be used in creating a self-extracting file, wherein the file is one of a plurality of file types; and

in response to only a single action, creating a self-extracting file from the input file, wherein the input file is configured to be automatically launched upon execution of the self-extracting file."

An embodiment of "receiving an input file to be used in creating a self-extracting file" is described at FIG. 4, 402; and P: 16; L: 22-23: "Beginning at a start state 400, the creation process proceeds to the next state 402, wherein an input file 202 is received.

An embodiment of "wherein the file is one of a plurality of file types" is described at FIG. 4 and P: 17; L: 1-15: "The result is that every file is transformed into an executable file, regardless of the file type of the received input file 202. Thus, for example, a bitmap (i.e., a '.bmp' file) received as an input file 202 is automatically transformed into a self extracting file (i.e., an '.exe' file) just as a text file (i.e., a '.txt' file) received as an input file 202 is also automatically transformed into a self-extracting file (i.e., an '.exe' file)."

An embodiment of "in response to only a single action" is described at P: 20; L: 2-4: "Once the name of the file to be converted into a universal self-extracting file appears in text window 502 [FIG. 5], the user performs a single action, such as clicking a 'Create File' button, to create the universal self-extracting file 204."

An embodiment of "creating a self-extracting file from the input file" is described at FIG. 4 404 – 420, and at P: 16; L: 20 – P: 19; L: 26.

An embodiment of "wherein the input file is configured to be automatically launched upon execution of the self-extracting file" is described at FIG. 4 412 and at P: 18; L: 4-6: "A loader is attached to the universal self-extracting file 204, which does not yet contain the archive header 208 or compressed input file data 206, in state 412 to automatically launch the input file 202 upon execution of the self-extracting file." An embodiment is further described at FIG. 6 and at P: 21; L: 8-11: "After decompression of the file to its original non-compressed form, the file is launched in state 608 with the appropriate application software for the given file. In on embodiment, for example, this is accomplished by calling the appropriate 'OPEN WITH' application registered with the current operating system."

Claim 4

Claim 4 recites "The method of claim 3, wherein the single action is a single click with a computer pointing device."

An embodiment of "wherein the single action is a single click" is found in original claim 4. Moreover, "wherein the single action is a single click" is described at P: 20; L: 2-4: "Once the name of the file to be converted into a universal self-extracting file appears in text window 502 [FIG. 5], the user performs a single action, such as clicking a 'Create File' button, to create the universal self-extracting file 204."

Embodiments of "with a computer pointing device" are inherent.

Claim 5

Claim 5 recites "The method of claim 3, wherein the single action is a double-click with a computer pointing device."

An embodiment of "wherein the single action is a double-click" is found in original claim 5.

Embodiments of “with a computer pointing device” are inherent.

Claim 6

Claim 6 recites “The method of claim 3, wherein the single action is speaking a sound.”

An embodiment of “wherein the single action is speaking a sound” is found in original claim 6.

Claim 7

Claim 7 recites “The method of claim 3, wherein the single action is pressing a key.”

An embodiment of “wherein the single action is pressing a key” is found in original claim 7.

Claim 8

Claim 8 recites: “The method of claim 3, wherein the single action is a call from a software routine.”

An embodiment of “wherein the single action is a call from a software routine” is found in original claim 8.

Claim 9

Claim 9 recites: “The method of claim 3, further comprising generating a filename for the self-extracting file, wherein the generated filename is based on a filename associated with the input file.”

An embodiment of “further comprising generating a filename for the self-extracting file, wherein the generated filename is based on a filename associated with the input file” is described at P: 16; L: 29 – P: 17; L: 3: “In one embodiment, the name is

generated based on the name and/or file type of the original input file 202 received by the compression module 108. For example, the name of the universal self-extracting file 204 may be generated based on the following pattern: FILENAME_FILEEXTENSION.EXE. Thus if the input file 202 is named 'house.bmp,' the produced universal self-extracting file 204 would be named 'house_bmp.exe.'" An embodiment is further described at FIG. 5 and P: 22; L: 27-31: "After receving the file, the universal self-extracting systems and methods use the filename of the digital photograph to create a name for an output file [that] is opened (i.e., created) and named 'George's Birthday_Jpeg_Lipon.exe.'"

Claim 10

Claim 10 recites "A method for creating a self-extracting file, the method comprising:

receiving a user selection of an input file to be used in creating a self-extracting file, wherein the input file is of any file type; and

automatically creating a self-extracting file configured to automatically launch the received input file responsive to execution of the self-extracting file."

An embodiment of "receiving a user selection of an input file to be used in creating a self-extracting file" is described at FIG. 4, 402; and P: 16; L: 22-23: "Beginning at a start state 400, the creation process proceeds to the next state 402, wherein an input file 202 is received; and further described at P: 20; L: 2-4: "Once the name of the file to be converted into a universal self-extracting file appears in text window 502, the user performs a single action, such as clicking a 'Create File' button, to create the universal self-extracting file 204."

An embodiment of "wherein the input file is of any file type" is described at FIG. 4 and P: 17; L: 1-15: "The result is that every file is transformed into an executable file, regardless of the file type of the received input file 202. Thus, for example, a bitmap (i.e., a '.bmp' file) received as an input file 202 is automatically transformed into a self

extracting file (i.e., an '.exe' file) just as a text file (i.e., a '.txt' file) received as an input file 202 is also automatically transformed into a self-extracting file (i.e., an '.exe' file)."

An embodiment of "automatically creating a self-extracting file" is described at FIG. 4 404 – 420, and at P:16; L: 20 – P: 19; L:26.

An embodiment of "configured to automatically launch the received input file responsive to execution of the self-extracting file" is described at FIG. 4 412 and at P: 18; L: 4-6: "A loader is attached to the universal self-extracting file 204, which does not yet contain the archive header 208 or compressed input file data 206, in state 412 to automatically launch the input file 202 upon execution of the self-extracting file." An embodiment is further described at FIG. 6 and at P: 21; L: 8-11: "After decompression of the file to its original non-compressed form, the file is launched in state 608 with the appropriate application software for the given file. In on embodiment, for example, this is accomplished by calling the appropriate 'OPEN WITH' application registered with the current operating system."

Claim 11

Claim 11 recites:

"The method of claim 10, wherein the creation of the self-extracting file comprises:

opening an output file;

attaching a decompression engine to the output file, wherein the decompression engine is capable of decompressing compressed data to a temporary file;

attaching a loader to the output file, wherein the loader configures the output file so as to automatically launch the temporary file after execution of the self-extracting file;

compressing the received input file according to a data compression method;

attaching an archive header including information about the compressed input file; and

closing the output file, wherein the closed output file is the self-extracting file."

An embodiment of "opening an output file" is described at FIG. 4 404, and at P: 16; L: 23-26. : "In state 404, the universal self-extracting file 204, which does not yet include the universal self-extracting stub portion 210, archive header portion 208, or compressed input file data portion 206, is initially opened (i.e., created) and named.

An embodiment of "attaching a decompression engine to the output file," is described at FIG. 4 408, and at P: 17; L: 7–10: "In states 406 and 408, the universal self-extracting file 204, which does not yet include the archive header portion 208 or compressed input file data portion 206, is configured as a self-extracting file by attaching a decompression engine and executable code to the file."

An embodiment of "wherein the decompression engine is capable of decompressing compressed data to a temporary file" is described at P: 17; L: 19-23: "The decompression engine, attached to the universal self-extracting file 204 in state 408, allows the compressed file data portion 206 to be decompressed without help from an external decompression utility. In one embodiment, the decompression engine decompresses the compressed file data portion 206 to a temporary file."

An embodiment of "attaching a loader to the output file, wherein the loader configures the output file so as to automatically launch the temporary file after execution of the self-extracting file" is described at FIG. 4 412, and at P: 18; L: 4-6: "A loader is attached to the universal self-extracting file 204, which does not yet contain the archive header 208 or compressed input file data 206, in state 412 to automatically launch the input file 202 upon execution of the self-extracting file."

An embodiment of "compressing the received input file according to a data compression method" is described at FIG. 4 416, and at P: 19; L: 1-3: "In state 416, the input file 202 is compressed using a standard data compression method. In one embodiment, the compression is performed using one compression method, such as LZ77 or a variant of LZ77, regardless of file type."

An embodiment of "attaching an archive header including information about the compressed input file" is described at FIG. 4 414 and 418; at P: 18; L: 22-24: "In state 414, a dummy archive header, marking a file location is written to reserve space (i.e.,

memory is allocated as a placeholder) for the additional archive header information to be added to the universal self-extracting file 204 in state 418."; and at P: 19; L: 15-17: "After the input file data has been compressed, the dummy archive header is updated at the appropriate marked location with information about the compressed input file data. The resulting archive header is attached to the file in state 418."

An embodiment of "closing the output file, wherein the closed output file is the self-extracting file" is described at FIG. 4 420, and at P: 19; L: 21-23: "Finally, in state 420, the universal self-extracting file 204 is closed and returned to the user or software module invoking the method and the creation process 110 [FIG. 1] proceeds to an end state 422."

Claim 12

Claim 12 recites: "The method of claim 11, wherein the input file is received from a user enabled electronic device."

An embodiment of "wherein the input file is received from a user enabled electronic device" is described at FIG. 5 502 and at P: 19; L: 29-31: "In the exemplary screen 500 the user types or otherwise selects the name of a file to be converted into a universal self-extracting file 204 into a text window 502."

Claim 13

Claim 13 recites: "The method of claim 11, wherein the input file is received from a software routine."

An embodiment of "wherein the input file is received from a software routine" is described in original claim 13.

Claim 14

Claim 14 recites: "The method of claim 11, wherein the data compression method is the same method for all received input files."

An embodiment of "wherein the data compression method is the same method for all received input files" is described at P: 19; L: 2-3: "In one embodiment, the compression is performed using one compression method, such as LZ77 or a variant of LZ77, regardless of file type."

Claim 15

Claim 15 recites: "The method of claim 11, wherein the data compression method is determined based on the file type of the received input file."

An embodiment of "wherein the data compression method is determined based on the file type of the received input file" is described at P: 19; L: 9-10: "In other embodiments, the compression method used to compress the received input file 202 may vary depending on the type of input file 202 received."

Claim 16

Claim 16 recites: "The method of claim 11, wherein the loader attached to the output file depends on the file type of the input file."

An embodiment of "wherein the loader attached to the output file depends on the file type of the input file" is described at P: 18; L: 19-10: "In one embodiment, the loader attached depends on the file type of the input file 202."

Claim 17

Claim 17 recites: "The method of claim 11, wherein the loader automatically unloads the temporary file."

An embodiment of "wherein the loader automatically unloads the temporary file" is described at P: 18; L: 14-15: "In one embodiment, the loader attached also performs unload and cleanup processes on the temporary file."

Claim 18

Claim 18 recites: "The method claim 11, further comprising attaching an unloader to the output file to automatically unload the temporary file."

An embodiment of "further comprising attaching an unloader to the output file to automatically unload the temporary file" is described at P: 18; L: 14-15: "In one embodiment, the loader attached also performs unload and cleanup processes on the temporary file."

Claim 19

Claim 19 recites: "The method of claim 18, wherein the unloader performs cleanup processes on the temporary file."

An embodiment of "wherein the unloader performs cleanup processes on the temporary file" is described at P: 18; L: 14-15: "In one embodiment, the loader attached also performs unload and cleanup processes on the temporary file."

Claim 20

Claim 20 recites: "A method for creating an executable file, comprising:

in response to only a single action, creating a self-extracting file from an input file, wherein the input file is one of a plurality of file types; and

automatically selecting a loader based on the input file's type; and

wherein the input file will be automatically launched upon execution of the self-extracting file."

An embodiment of "in response to only a single action," is described at P: 20; L: 2-4: "Once the name of the file to be converted into a universal self-extracting file appears in text window 502 [FIG. 5], the user performs a single action, such as clicking a 'Create File' button, to create the universal self-extracting file 204."

An embodiment of "creating a self-extracting file from an input file," is described at FIG. 4 404 – 420, and at P: 16; L: 20 – P: 19; L: 26.

An embodiment of "wherein the input file is one of a plurality of file types" is described at FIG. 4 and P: 17; L: 1-15: "The result is that every file is transformed into an executable file, regardless of the file type of the received input file 202. Thus, for example, a bitmap (i.e., a '.bmp' file) received as an input file 202 is automatically transformed into a self extracting file (i.e., an '.exe' file) just as a text file (i.e., a '.txt' file) received as an input file 202 is also automatically transformed into a self-extracting file (i.e., an '.exe' file)."

An embodiment of "automatically selecting a loader based on the input file's type" is described at P: 18; L: 9-10: "In one embodiment, the loader attached depends on the file type of the input file 202."

An embodiment of "wherein the input file will be automatically launched upon execution of the self-extracting file" is described at FIG. 4 412 and at P: 18; L: 4-6: "A loader is attached to the universal self-extracting file 204, which does not yet contain the archive header 208 or compressed input file data 206, in state 412 to automatically launch the input file 202 upon execution of the self-extracting file." An embodiment is further described at FIG. 6 608 and at P: 21; L: 8-11: "After decompression of the file to its original non-compressed form, the file is launched in state 608 with the appropriate application software for the given file. In on embodiment, for example, this is accomplished by calling the appropriate 'OPEN WITH' application registered with the current operating system."

Claim 21

Claim 21 recites: "A method of creating a self-extracting file comprising:

displaying a first frame used to allow a user to specify an input file to be converted to a self-extracting file;

receiving the input file specified by the user, wherein the received input file is automatically configured as a self-extracting file, and wherein the input file is automatically launched upon execution of the self-extracting file; and

displaying a second frame, wherein the second frame includes a link related to the self-extracting file created from the user specified input file."

An embodiment of "displaying a first frame used to allow a user to specify an input file to be converted to a self-extracting file" is described at P: 20; L: 2-4: "Once the name of the file to be converted into a universal self-extracting file appears in text window 502 [FIG. 5]..."; at FIG. 5 502; and at P: 19; L: 29-31: "In the exemplary screen 500 the user types or otherwise selects the name of a file to be converted into a universal self-extracting file 204 into a text window 502."

An embodiment of "receiving the input file specified by the user" is described at P: 20; L: 2-4: "the user performs a single action, such as clicking a 'Create File' button, to create the universal self-extracting file 204."

An embodiment of "wherein the received input file is automatically configured as a self-extracting file" is described at FIG. 4 404-420, and at P: 16; L: 20 – P: 19; L:26.

An embodiment of "wherein the input file is automatically launched upon execution of the self-extracting file" is described at FIG. 4 412 and at P: 18; L: 4-6: "A loader is attached to the universal self-extracting file 204, which does not yet contain the archive header 208 or compressed input file data 206, in state 412 to automatically launch the input file 202 upon execution of the self-extracting file." An embodiment is further described at FIG. 6 and at P: 21; L: 8-11: "After decompression of the file to its original non-compressed form, the file is launched in state 608 with the appropriate application software for the given file. In on embodiment, for example, this is accomplished by calling the appropriate 'OPEN WITH' application registered with the current operating system."

An embodiment of "displaying a second frame, wherein the second frame includes a link related to the self-extracting file created from the user specified input file" is described at FIG. 5 504 and at P: 19; L: 12-14: "In one embodiment, the creation process 110 operates sequentially on each of the three input files, and the resulting names of the three universal self-extracting files appears in text window 504..."

Claim 22

Claim 22 recites: "A system for creating a self-extracting file comprising:

a receiving module configured to receive an input file, wherein the input file received is one of a plurality of file types and wherein the input file includes an

associated filename;

a naming module configured to create and name an output file, wherein the output filename is generated from the associated filename of the input file and wherein the naming module receives the input file from the receiving module;

a self-extracting module configured to transform the output file into a executable file, wherein the self-extracting module receives the input file and the output file from the naming module;

a loader module configured to setup the executable file to launch the input file upon execution of the executable file, wherein the loader module receives the executable file and the input file from the self-extracting module; and

a compressing module configured to compress the input file and attach the compressed input file to the executable file, wherein the compressing module receives the input file and the executable file from the loader module;

wherein each module is embodied in hardware, in firmware, or in a collection of software instructions stored in a tangible computer-readable medium."

An embodiment of "a receiving module configured to receive an input file, wherein the input file received is one of a plurality of file types and wherein the input file includes an associated filename" is described at FIG. 1 101 and at P: 10; L: 20: "The receiving module 101 receives an input file 202 (FIGURE 2) for use in creating a universal self-extracting file 204."

An embodiment of "a naming module configured to create and name an output file, wherein the output filename is generated from the associated filename of the input file and wherein the naming module receives the input file from the receiving module" is described at FIG. 1 102; at P: 10; L: 26-27: "The naming module 102 uses the received input file 202 to begin creating a universal self-extracting file 204."; and at P: 10; L: 29-30: "The naming module 102 generates a name for the universal self-extracting file 204."

An embodiment of "a self-extracting module configured to transform the output file into a executable file, wherein the self-extracting module receives the input file and the output file from the naming module" is described at FIG. 1 104 and at P: 11; L: 29-

31: "The self-extracting module 104 receives the named universal self-extracting file 204 from the naming module 102 and attaches executable code and a decompression engine as a stub, making the resulting file a self-extracting file."

An embodiment of "a loader module configured to setup the executable file to launch the input file upon execution of the executable file, wherein the loader module receives the executable file and the input file from the self-extracting module" is described at FIG. 1 106 and at P: 12; L: 18-21: "The loader module 106 receives the file, which includes the executable code and the attached decompression engine, from the self-extracting module 104 and configures the file to be automatically launched after decompression of the compressed input file data by the attached decompression engine."

An embodiment of "a compressing module configured to compress the input file and attach the compressed input file to the executable file, wherein the compressing module receives the input file and the executable file from the loader module" is described at FIG. 1 108; at P: 13; L: 19-23: "The compression module 108 receives the universal self-extracting file 204, which includes the universal self-extracting stub portion 210 and the input file data to be compressed, from the loader module 106. The compression module 108 applies a compression method to the input file 202 to produce a compressed input file data portion 206."; and at P: 14; L: 3-4: "After the input file data has been compressed, the compression module 108 attaches an archive header portion 208 to the compressed input file data portion 206."

An embodiment of "wherein each module is embodied in hardware, in firmware, or in a collection of software instructions stored in a tangible computer-readable medium" is described at P: 9; L: 11-13: "As used herein, the word module refers to logic embodied in hardware or firmware, or to a collection of software instructions, possibly having entry and exit points, written in a programming language, such as, for example C++." and at P: 9; L: 18: "Software instructions may be embedded in firmware, such as an EPROM or EEPROM." Other computer-readable media carrying computer readable instructions are inherent.

Claim 23

Claim 23 recites: "The system of claim 22, wherein the loader module is further configured to setup the executable file to perform unload processes."

An embodiment of "wherein the loader module is further configured to setup the executable file to perform unload processes" is described at P: 13; L: 8-11: "In those cases where cleanup is needed, such as, for example, when the input file 202 is a DLL file, the loader module 106 configures the universal self-extracting file 204 to unload and clean-up the temporary extracted DLL file after the user is finished using the file."

Claim 24

Claim 24 recites: "A system for creating, in response to a single action, a self-extracting file from an associated input file, wherein the associated input file is automatically launched upon execution of the self-extracting file, and wherein a user is not required to separately choose a data compression method, create a compressed archive using the chosen compression method, select an input file to be launched upon decompression of the compressed archive, and create a self-extracting file from the compressed archive, the system comprising:

a first module for receiving a user selection of an input file to be compressed, wherein the input file is one of a plurality of file types;

a second module for compressing the received input file according to a data compression method; and

a third module for creating, in response to only a single action by a user, an executable file from the compressed input file, wherein the input file will be automatically launched upon execution of the executable fie."

An embodiment of "a first module for receiving a user selection of an input file to be compressed" is described at FIG. 1 101 and at P: 10; L: 20: "The receiving module 101 receives an input file 202 (FIGURE 2) for use in creating a universal self-extracting file 204."

An embodiment of "wherein the input file is one of a plurality of file types" is

described at FIG. 4 and P: 17; L: 1-15: "The result is that every file is transformed into an executable file, regardless of the file type of the received input file 202. Thus, for example, a bitmap (i.e., a '.bmp' file) received as an input file 202 is automatically transformed into a self extracting file (i.e., an '.exe' file) just as a text file (i.e., a '.txt' file) received as an input file 202 is also automatically transformed into a self-extracting file (i.e., an '.exe' file)."

An embodiment of "a second module for compressing the received input file according to a data compression method" is described at FIG. 1 108 and at P: 13; L: 19-23: "The compression module 108 receives the universal self-extracting file 204, which includes the universal self-extracting stub portion 210 and the input file data to be compressed, from the loader module 106. The compression module 108 applies a compression method to the input file 202 to produce a compressed input file data portion 206."

An embodiment of "a third module for creating, in response to only a single action by a user, an executable file from the compressed input file" is described at P: 20; L: 2-4: "the user performs a single action, such as clicking a 'Create File' button, to create the universal self-extracting file 204." and at FIG. 1 104 and at P: 11; L: 29-31: "The self-extracting module 104 receives the named universal self-extracting file 204 from the naming module 102 and attaches executable code and a decompression engine as a stub, making the resulting file a self-extracting file."

An embodiment of "wherein the input file will be automatically launched upon execution of the executable fie" is described at FIG. 1 106 and at P: 12; L: 18-21: "The loader module 106 receives the file, which includes the executable code and the attached decompression engine, from the self-extracting module 104 and configures the file to be automatically launched after decompression of the compressed input file data by the attached decompression engine."

Claim 25

Claim 25 recites: "A tangible computer-readable medium carrying computer-executable instructions configured to cause a computer to:

provide a compressed input data portion corresponding to an input data file, the compressed input data portion including data compressed according to a preselected data compression method;

provide an archive header portion, wherein the archive header portion includes information about the compressed input data portion; and

provide a self-extracting stub portion, wherein the self-extracting stub portion is automatically attached to the compressed input data portion and the archive header portion, and wherein the self-extracting stub portion includes

a decompression engine to decompress the compressed input data portion, and

a loader operable to launch the decompressed input data portion with appropriate application software for handling the input data file."

An embodiment of "provide a compressed input data portion corresponding to an input data file" is described at FIG. 2 206 and at P: 14; L: 14-16: "Given an input file 202, the systems and methods produce a universal self-extracting file 204, which includes a compressed input file data portion 206, a archive header portion 208, and a universal self-extracting stub portion 210"; and at FIG. 4 416 and at P: 19; L: 1-2: "In state 416, the input file 202 is compressed using a standard data compression method."

An embodiment of "the compressed input data portion including data compressed according to a preselected data compression method" is described at P: 19; L: 3-4: "In this embodiment, the compression method used to compress the input file data is pre-determined..."

An embodiment of "provide an archive header portion, wherein the archive header portion includes information about the compressed input data portion" is described at FIG. 2 208 and at P: 14; L: 14-16: "Given an input file 202, the systems and methods produce a universal self-extracting file 204, which includes a compressed input file data portion 206, a archive header portion 208, and a universal self-extracting stub portion 210"; at FIG. 4 414 and 418; at P: 18; L: 22-24: "In state 414, a dummy archive header, marking a file location is written to reserve space (i.e., memory is

allocated as a placeholder) for the additional archive header information to be added to the universal self-extracting file 204 in state 418."; and at P: 19; L: 15-17: "After the input file data has been compressed, the dummy archive header is updated at the appropriate marked location with information about the compressed input file data. The resulting archive header is attached to the file in state 418."

An embodiment of "provide a self-extracting stub portion" is described at FIG. 2 210 and at P: 14; L: 14-16: Given an input file 202, the systems and methods produce a universal self-extracting file 204, which includes a compressed input file data portion 206, a archive header portion 208, and a universal self-extracting stub portion 210"; and P: 14; L: 19-22: "UNIVERSAL SELF-EXTRACTING STUB PORTION (EXECUTABLE CODE + DECOMPRESSION ENGINGE [sic] + LOADER) + ARCHIVE HEADER PORTION + COMPRESSED INPUT FILE DATA PORTION = UNIVERSAL SELF-EXTRACTING FILE."; at FIG. 4 406, 408, 412; and at P: 17; L: 7-10: "In states 406 and 408, the universal self-extracting file 204, which does not yet include the archive header portion 208 or compressed input file data portion 206, is configured as a self-extracting file by attaching a decompression engine and executable code to the file,"; and at P: 18; L: 4-6: "A loader is attached to the universal self-extracting file 204, which does not yet contain the archive header 208 or compressed input file data 206, in state 412 to automatically launch the input file 202 upon execution of the self-extracting file."

An embodiment of "wherein the self-extracting stub portion is automatically attached to the compressed input data portion and the archive header portion" is described at FIG. 4 414, 416, and 418 and at P: 18; L: 22-24: "In state 414, a dummy archive header, marking a file location is written to reserve space (i.e., memory is allocated as a placeholder) for the additional archive header information to be added to the universal self-extracting file 204 in state 418,"; at P: 19; L: 15-17: "After the input file data has been compressed, the dummy archive header is updated...The resulting archive header is attached to the file in state 418."; and at P: 19; L: 6-8: "In one embodiment, the compressed data is written directly to the open universal self-extracting file 204, which was opened in state 404."

An embodiment of "wherein the self-extracting stub portion includes a decompression engine to decompress the compressed input data portion" is described

at P: 14; L: 19-22: "UNIVERSAL SELF-EXTRACTING STUB PORTION (EXECUTABLE CODE + DECOMPRESSION ENGINGE [sic] + LOADER)"

An embodiment of "wherein the self-extracting stub portion includes a loader operable to launch the decompressed input data portion with appropriate application software for handling the input data file" is described at FIG. 4 412, and at P: 18; L: 4-6: "A loader is attached to the universal self-extracting file 204, which does not yet contain the archive header 208 or compressed input file data 206, in state 412 to automatically launch the input file 202 upon execution of the self-extracting file."

Claim 26

Claim 26 recites: "A method for creating, in response to a single action, a self-extracting file, the method comprising:

receiving an input file from a user to be used in creating a self-extracting file, wherein the input file is of any file type; and

automatically creating a self-extracting file."

An embodiment of "receiving an input file from a user to be used in creating a self-extracting file" is described at P: 20; L: 2-4: "the user performs a single action, such as clicking a 'Create File' button, to create the universal self-extracting file 204."

An embodiment of "wherein the input file is of any file type" is described at FIG. 4 and P: 17; L: 1-15: "The result is that every file is transformed into an executable file, regardless of the file type of the received input file 202. Thus, for example, a bitmap (i.e., a '.bmp' file) received as an input file 202 is automatically transformed into a self extracting file (i.e., an '.exe' file) just as a text file (i.e., a '.txt' file) received as an input file 202 is also automatically transformed into a self-extracting file (i.e., an '.exe' file)."

An embodiment of "automatically creating a self-extracting file" is described at FIG. 4 404 – 420, and at P: 16; L: 20 – P: 19; L: 26.

Claim 27

Claim 27 recites: "A method for creating an executable file, the method comprising:

receiving, in response to a single action, an input file to be used in creating an executable file, wherein the input file is one of a plurality of file types; and

without further instructions, creating an executable file using the received input file, wherein the executable file includes a compressed copy of the input file, and wherein the compressed copy of the input file is automatically decompressed and launched upon execution of the executable file."

An embodiment of "receiving, in response to a single action, an input file to be used in creating an executable file" is described at P: 20; L: 2-4: "the user performs a single action, such as clicking a 'Create File' button, to create the universal self-extracting file 204,"; at FIG. 4, 402; and P: 16; L: 22-23: "Beginning at a start state 400, the creation process proceeds to the next state 402, wherein an input file 202 is received."

An embodiment of "wherein the input file is one of a plurality of file types" is described at FIG. 4 and P: 17; L: 1-15: "The result is that every file is transformed into an executable file, regardless of the file type of the received input file 202. Thus, for example, a bitmap (i.e., a '.bmp' file) received as an input file 202 is automatically transformed into a self extracting file (i.e., an '.exe' file) just as a text file (i.e., a '.txt' file) received as an input file 202 is also automatically transformed into a self-extracting file (i.e., an '.exe' file)."

An embodiment of "without further instructions, creating an executable file using the received input file" is described at FIG. 4 404 – 420, and at P: 16; L: 20 – P: 19; L: 26.

An embodiment of "wherein the executable file includes a compressed copy of the input file" is described at FIG. 2 210 and at P: 14; L: 14-16: Given an input file 202, the systems and methods produce a universal self-extracting file 204, which includes a compressed input file data portion 206, a archive header portion 208, and a universal self-extracting stub portion 210"; and P: 14; L: 19-22: "UNIVERSAL SELF-EXTRACTING STUB PORTION (EXECUTABLE CODE + DECOMPRESSION ENGINGE [sic] + LOADER) + ARCHIVE HEADER PORTION + COMPRESSED INPUT FILE DATA PORTION = UNIVERSAL SELF-EXTRACTING FILE."

An embodiment of “wherein the compressed copy of the input file is automatically decompressed and launched upon execution of the executable file” is described at P: 7; L: 31 - P: 8; L: 3: "Using the input file, the methods produce, without needing further instruction, a self-extracting file that, upon execution, automatically extracts the input file data into a temporary file and launches the temporary file (i.e., the input file) with the appropriate application software for that file type."

Claim 28

Claim 28 recites: “A process for producing, in response to a single action, a computer file, the process comprising:

receiving an input file;

automatically opening an output file;

automatically adding a decompression engine to the output file for decompressing compressed data;

automatically adding loader code to the output file for launching the input file with the appropriate application software for handling the input file;

automatically adding an archive header to the output file, wherein the archive header includes information relating to the input file;

automatically compressing the input file according to a data compression method;

automatically updating the archive header to include information about the compressed input file; and

automatically closing the output file.”

An embodiment of “receiving an input file” is described at FIG. 4, 402; and P: 16; L: 22-23: “Beginning at a start state 400, the creation process proceeds to the next state 402, wherein an input file 202 is received; and further described at P: 20; L: 2-4: “Once the name of the file to be converted into a universal self-extracting file appears in text window 502, the user performs a single action, such as clicking a ‘Create File’ button, to create the universal self-extracting file 204.”

An embodiment of “automatically opening an output file” is described at FIG. 4

404, and at P: 16; L: 23-26: "In state 404, the universal self-extracting file 204, which does not yet include the universal self-extracting stub portion 210, archive header portion 208, or compressed input file data portion 206, is initially opened (i.e., created) and named.

An embodiment of "automatically adding a decompression engine to the output file for decompressing compressed data" is described at FIG. 4 408, and at P: 17; L: 7–10: "In states 406 and 408, the universal self-extracting file 204, which does not yet include the archive header portion 208 or compressed input file data portion 206, is configured as a self-extracting file by attaching a decompression engine and executable code to the file."

An embodiment of "automatically adding loader code to the output file for launching the input file with the appropriate application software for handling the input file" is described at FIG. 4 412, and at P: 18; L: 4-6: "A loader is attached to the universal self-extracting file 204, which does not yet contain the archive header 208 or compressed input file data 206, in state 412 to automatically launch the input file 202 upon execution of the self-extracting file."

An embodiment of "automatically adding an archive header to the output file, wherein the archive header includes information relating to the input file" is described at FIG. 4 414 and 418; at P: 18; L: 22-24: "In state 414, a dummy archive header, marking a file location is written to reserve space (i.e., memory is allocated as a placeholder) for the additional archive header information to be added to the universal self-extracting file 204 in state 418."

An embodiment of "automatically compressing the input file according to a data compression method" is described at FIG. 4 416, and at P: 19; L: 1-3: "In state 416, the input file 202 is compressed using a standard data compression method. In one embodiment, the compression is performed using one compression method, such as LZ77 or a variant of LZ77, regardless of file type."

An embodiment of "automatically updating the archive header to include information about the compressed input file" is described at P: 19; L: 15-17: "After the input file data has been compressed, the dummy archive header is updated at the appropriate marked location with information about the compressed input file data. The

resulting archive header is attached to the file in state 418."

An embodiment of "automatically closing the output file" is described at FIG. 4 420, and at P: 19; L: 21-23: "Finally, in state 420, the universal self-extracting file 204 is closed and returned to the user or software module invoking the method and the creation process 110 [FIG. 1] proceeds to an end state 422."

Claim 29

Claim 29 recites: "The product produced by the process of claim 28."

Embodiments of "The product produced by the process of claim 28" are inherent.

Claim 30

Claim 30 recites: "A method for creating an executable file, the method comprising:

in response to a single action, receiving an input file to be used in creating an executable file, wherein the input file is one of a plurality of file types; and

without further instruction, creating an executable file using the received input file, wherein the executable file comprises:

a compressed input data portion including data compressed according to a data compression method;

an archive header portion including information about the compressed input data portion; and

a stub portion, wherein the stub portion is automatically attached to the compressed input data portion and the archive header portion, and wherein the stub portion includes a decompression engine to decompress the compressed input data portion and a loader to launch the decompressed input data portion."

An embodiment of "in response to a single action, receiving an input file to be used in creating an executable file" is described at FIG. 4, 402; and P: 16; L: 22-23:

“Beginning at a start state 400, the creation process proceeds to the next state 402, wherein an input file 202 is received; and further described at P: 20; L: 2-4: “Once the name of the file to be converted into a universal self-extracting file appears in text window 502, the user performs a single action, such as clicking a ‘Create File’ button, to create the universal self-extracting file 204.”

An embodiment of “wherein the input file is one of a plurality of file types” is described at FIG. 4 and P: 17; L: 1-15: “The result is that every file is transformed into an executable file, regardless of the file type of the received input file 202. Thus, for example, a bitmap (i.e., a ‘.bmp’ file) received as an input file 202 is automatically transformed into a self extracting file (i.e., an ‘.exe’ file) just as a text file (i.e., a ‘.txt’ file) received as an input file 202 is also automatically transformed into a self-extracting file (i.e., an ‘.exe’ file).”

An embodiment of “without further instruction, creating an executable file using the received input file” is described at FIG. 4 404 – 420, and at P: 16; L: 20 – P: 19; L: 26.

An embodiment of “wherein the executable file comprises a compressed input data portion including data compressed according to a data compression method” is described at FIG. 2 206 and at P: 14; L: 14-16: “Given an input file 202, the systems and methods produce a universal self-extracting file 204, which includes a compressed input file data portion 206... ”

An embodiment of “wherein the executable file comprises an archive header portion including information about the compressed input data portion” is described at FIG. 2 208 and at P: 14; L: 14-16: “Given an input file 202, the systems and methods produce ... a archive header portion 208”

An embodiment of “wherein the executable file comprises a stub portion” is described at FIG. 2 210 and at P: 14; L: 14-16: “Given an input file 202, the systems and methods produce a universal self-extracting file 204, which includes ... a universal self-extracting stub portion 210...”

An embodiment of “wherein the stub portion is automatically attached to the compressed input data portion and the archive header portion” is described at FIG. 4 414, 416, and 418 and at P: 18; L: 22-24: "In state 414, a dummy archive header,

marking a file location is written to reserve space (i.e., memory is allocated as a placeholder) for the additional archive header information to be added to the universal self-extracting file 204 in state 418,"; at P: 19; L: 15-17: "After the input file data has been compressed, the dummy archive header is updated...The resulting archive header is attached to the file in state 418."; and at P: 19; L: 6-8: "In one embodiment, the compressed data is written directly to the open universal self-extracting file 204, which was opened in state 404."

An embodiment of "wherein the stub portion includes a decompression engine to decompress the compressed input data portion" is described at P: 14; L: 19-22: "UNIVERSAL SELF-EXTRACTING STUB PORTION (EXECUTABLE CODE + DECOMPRESSION ENGINGE [sic] + LOADER)."

An embodiment of "wherein the stub portion includes a loader to launch the decompressed input data portion" is described at FIG. 4 412, and at P: 18; L: 4-6: "A loader is attached to the universal self-extracting file 204, which does not yet contain the archive header 208 or compressed input file data 206, in state 412 to automatically launch the input file 202 upon execution of the self-extracting file."

Claim 31

Claim 31 recites: "A method for using an executable file, the method comprising:

in response to a first action, creating an executable file from an input file, wherein the executable file includes a compressed copy of the input file, and wherein the executable file includes code to decompress and to load the compressed input file; and

in response to a second action, executing the executable file to decompress the compressed copy of the input file and launching the decompressed input file with appropriate application software."

An embodiment of "in response to a first action, creating an executable file from an input file" is described at FIG. 4 404 – 420, and at P: 16; L: 20 – P: 19; L: 26.

An embodiment of "wherein the executable file includes a compressed copy of the input file" is described at FIG. 2 206 and at P: 14; L: 14-16: "Given an input file 202,

the systems and methods produce a universal self-extracting file 204, which includes a compressed input file data portion 206... "

An embodiment of "wherein the executable file includes code to decompress and to load the compressed input file" is described at P: 17; L: 19-23: "The decompression engine, attached to the universal self-extracting file 204 in state 408, allows the compressed file data portion 206 to be decompressed without help from an external decompression utility. In one embodiment, the decompression engine decompresses the compressed file data portion 206 to a temporary file,"; and at FIG. 4 412, and at P: 18; L: 4-6: "A loader is attached to the universal self-extracting file 204, which does not yet contain the archive header 208 or compressed input file data 206, in state 412 to automatically launch the input file 202 upon execution of the self-extracting file."

An embodiment of "in response to a second action, executing the executable file to decompress the compressed copy of the input file" is described at FIG. 6 602, 604, and 606 and at P: 20; L: 28 - P: 21; L: 6: "Beginning at a start state 600, the universal self-extracting file 204 is executed in state 602 when the executable code of the universal self-extracting stub portion 210 is activated. The universal self-extracting file 204 may be executed by a user selecting the file (e.g., double clicking on the file, pressing enter, etc.)...Upon execution of the universal self-extracting file 204 in state 602, the decompression engine is called in state 604. The decompression engine, in state 606, decompresses the compressed input file data portion 206 into a temporary file."

An embodiment of "launching the decompressed input file with appropriate application software" is described at FIG. 6 608 and at P: 21; L: 8-9: "After decompression of the file to its original non-compressed form, the file is launched in state 608 with the appropriate application software for the given file. "

Claim 32

Claim 32 recites: "A method for creating a self-extracting file, the method comprising:

receiving, in response to a single action, an input file to be used in creating a self-extracting file;

without further instruction, creating a self-extracting file using the input file and automatically launching the input file upon execution of the self-extracting file."

An embodiment of "receiving, in response to a single action, an input file to be used in creating a self-extracting file" is described at FIG. 4, 402; and P: 16; L: 22-23: "Beginning at a start state 400, the creation process proceeds to the next state 402, wherein an input file 202 is received; and further described at P: 20; L: 2-4: "Once the name of the file to be converted into a universal self-extracting file appears in text window 502, the user performs a single action, such as clicking a 'Create File' button, to create the universal self-extracting file 204."

An embodiment of "without further instruction, creating a self-extracting file using the input file" is described at FIG. 4 404 – 420, and at P: 16; L: 20 – P: 19; L: 26.

An embodiment of "automatically launching the input file upon execution of the self-extracting file" is described at FIG. 6 608 and at P: 21; L: 8-9: "After decompression of the file to its original non-compressed form, the file is launched in state 608 with the appropriate application software for the given file. "

Claim 33

Claim 33 recites: "The method of claim 32, wherein the input file is an executable routine and wherein a function of the executable routine is called upon loading of the executable routine."

An embodiment of "wherein the input file is an executable routine and wherein a function of the executable routine is called upon loading of the executable routine" is described at P: 21; L: 15-19: "In another embodiment (not shown) dealing with dynamically loaded executable modules, such as with DLL files, the decompress/launching process may load the temporary DLL into memory and call the

DLL's exported function. For example, a DLL may be loaded into memory, its function called, and the DLL may then pop up a user interface for an application. ”

Claim 34

Claim 34 recites: “The method of claim 32, wherein the input file is a dynamic link library file.”

An embodiment of “wherein the input file is a dynamic link library file” is described at P: 12; L: 28-30: “For example, in the case that the input file 202 is a Dynamic Link Library (DLL) file, the loader module 106 may use script codes to cause the DLL's function to be called after execution of the universal self-extracting file 204”

(H) Grounds of rejection to be reviewed on appeal

1. Claim 25 is rejected under 35 U.S.C. § 101 because invention is directed to non-statutory subject matter.

2. Claims 1-10, 20, 24, 26, 27, and 31-33 are rejected under 35 U.S.C § 102(e) as being anticipated by Halpern et al. [hereafter Halpern], U.S. Patent No. 6,282,711, filed on Aug. 10, 1999.

3. Claims 21-23 are rejected under 35 U.S.C § 102(e) as being anticipated by Wygodny et al. [hereafter Wygodny], U.S. Patent No. 6,202,199 based on provisional application No. 60/055,165 filed on July 31, 1997.

4. Claims 11-19, 25, 28-29, 30, and 34 are rejected under 35 U.S.C. § 103(a) as being unpatentable over Halpern et al. [hereafter Halpern], U.S. Patent No. 6,282,711 filed on Aug. 10, 1999 as applied to claims 10 and 32, above, and further in view of Gage et al. [hereafter Gage], U.S. Patent No. 5,923,846 published on July 13, 1999.

(I) Argument

In the Notice of Panel Decision mailed 1/12/2010, only claims 1-10, 20, 24, 26, 27, and 31-33 are listed as rejected. However, claims 1-34 are pending in the application, and claims 1-34 were listed as rejected in the Advisory Action mailed 9/14/2009 and earlier communications. A telephone call was placed and voicemail left for the Examiner on 2/8/2010 to request clarification, but no response was received.

The Applicant's agent herein provides reasons for allowability for all of pending claims 1-34.

Claim 25 is rejected under 35 U.S.C. § 101 because invention is directed to non-statutory subject matter.

Claim 25 was previously amended to overcome the 35 U.S.C. §101 rejection. The Examiner entered the amendment but did not reply as to whether the amendment overcame the section 101 rejection. The Applicant's agent believes claim 25 is now allowable in view of 35 U.S.C. §101.

The Board is respectfully requested to remove the 35 U.S.C. §101 rejection of claim 25 and rule claim 25 allowable.

Claims 1-10, 20, 24, 26, 27, and 31-33 are rejected under 35 U.S.C § 102(e) as being anticipated by Halpern et al. [hereafter Halpern], U.S. Patent No. 6,282,711, filed on Aug. 10, 1999.

Claim 1

Claim 1 recites "receiving from the user enabled electronic device, an input file to be used in creating a self-extracting file; and without further action by the user enabled electronic device, creating a self-extracting file using the input file, wherein the input file is configured to be automatically launched upon execution of the self-extracting file."

Halpern does not disclose receiving an input file to be used in creating a self-extracting file from a user-enabled electronic device.

Contrary to the Examiner's contention, Halpern [C:5; L: 41-44], does not disclose receiving an input file from a user-enabled electronic device. Rather, Halpern discloses providing one of three selected user interfaces, UI-1, UI-2, or UI-3 to the user for selecting application components and options, followed by subsequent delivery of a file to the user by a server.

Even if one (incorrectly) equates selection an input file through Halpern's components and options interface to receiving an input file from a user-enabled electronic device (which is not shown in Halpern), Halpern still would not disclose this limitation because Halpern's components and options do not have a 1:1 correspondence with files.

Following Halpern's selection of components and options, an "options manager" retrieves metadata from a database and an "installer set generator" accesses a component pool to produce a custom set of files corresponding to the user's selections [C: 7; L: 23-38]. Thus, the selected components and options are program functionalities that are used as input to the options manager, and do not represent files.

Accordingly, Halpern does not disclose receiving an input file from the user enabled electronic device, as recited by claim 1.

Halpern does not disclose creating a self-extracting file using the input file without further action by the user enabled electronic device.

Even if user selection of "components and options" is incorrectly assumed equivalent to receiving an input file, Halpern is, at best, silent regarding further action. Helpern does not disclose whether or not further action is required by the user, or by the user enabled electronic device.

Halpern does not disclose the input file is configured to be automatically launched upon execution of the self-extracting file.

Even if one incorrectly assumes Halpern discloses receiving an input file from a user-enabled electronic device, and assumes that no further action is required by the user-enabled electronic device, Halpern still does not disclose that the input file is configured to be automatically launched upon execution of the self-extracting file.

Halpern discloses that the compressed files are configured to be automatically extracted. [emphasis added] Automatic extraction of a file is not the same as automatically launching the file after it is extracted.

Moreover, by comparing Halpern's Step 7 to Step 12 [C: 7; L: 55-56], it is apparent that Halpern's "*input file*" is not even automatically *installed* upon execution of the self-extracting file. The client installer program is a separate entity, as evidenced by Halpern's indication that "the user executes the retrieved file and runs the client installer." [emphasis added] A program that is not even installed automatically after extraction also cannot be launched automatically after extraction, as recited by claim 1. This view is further strengthened by reference to Halpern [C: 6; L: 47-52], which contains its own rebuttal of the Examiner's position: "the user may simply execute the received setup.exe or install.exe file to immediately install the applications and options…" Requiring a user to execute a program (setup.exe or install.exe) is inconsistent with automatic launching upon execution of the self-extracting file (even if it is an input file received from the user enabled electronic device that is launched).

With respect to the Examiner's contention that Halpern discloses "auto-start", [C: 3; L: 42-49; and Fig. 1] there is no disclosure of anything that bears any relation to the recited limitations of claim 1. Halpern does not mention anything that could remotely be interpreted as "auto-start" at the cited location. The Applicant's agent searched the entirety of Halpern and found that there is a mention of an "auto-start utility" [C: 6; L: 49], but this is not related to the recitation of claim 1. Halpern apparently only includes this "auto-start utility" in a list of possible program components and does not describe what it is. It is suggested that perhaps this relates to a program component that can automatically launch an application responsive to clicking on a linked data file type. But this does not mean that the compressed file is automatically launched upon extraction. Nowhere does Halpern disclose an input file that is "configured automatically launched upon execution of the self-extracting file," as recited by claim 1 and, as explained above, implies the opposite.

Generally, the Examiner seems to not give proper weight to the claim language "wherein the input file is configured to be automatically launched upon execution of the

self-extracting file." A common thread through the citations provided by the Examiner is that extraction will occur automatically upon execution of a self-extracting compressed file. In contrast, none of the cited art discloses or reasonably suggests that an input file is not only extracted, but also launched for execution upon extraction.

Accordingly, Halpern does not disclose all the limitations of claim 1, and claim 1 is allowable over Halpern.

Claim 2

Claim 2 recites automatically generating a filename for the self-extracting based in part on the filename of the received input file.

Claim 2 is allowable by virtue of its dependency from claim 1.

Moreover, Halpern does not disclose naming a self-extracting file based in part on the filename of a received input file (even if, *arguendo*, the input file was received from the user enabled electronic device). Halpern is apparently silent on the names of input files, and also does not disclose naming a self-extracting file based in part on the name of an input file.

Claim 2 is also allowable over Halpern for at least this additional reason.

Claim 3

Claim 3 recites "receiving an input file to be used in creating a self-extracting file, wherein the file is one of a plurality of file types; and in response to only a single action, creating a self-extracting file from the input file, wherein the input file is configured to be automatically launched upon execution of the self-extracting file."

Claim 3 is allowable for reasons similar to those given for claim 1. Namely, Halpern does not disclose the input file is configured to be automatically launched upon execution of the self-extracting file.

Halpern discloses that the compressed files are configured to be automatically extracted. [emphasis added] Automatic extraction of a file is not the same as automatically launching the file after it is extracted.

Moreover, by comparing Halpern's Step 7 to Step 12 [C: 7; L: 55-56], it is apparent that Halpern's "*input file*" is not even automatically *installed* upon execution of

the self-extracting file. The client installer program is a separate entity, as evidenced by Halpern's indication that "the user executes the retrieved file and runs the client installer." [emphasis added] A program that is not even installed automatically after extraction also cannot be launched automatically after extraction, as recited by claim 1. This view is further strengthened by reference to Halpern [C: 6; L: 47-52], which contains its own rebuttal of the Examiner's position: "the user may simply execute the received setup.exe or install.exe file to immediately install the applications and options…" Requiring a user to execute a program (setup.exe or install.exe) is inconsistent with automatic launching upon execution of the self-extracting file (even if it is an input file received from the user enabled electronic device that is launched).

With respect to the Examiner's contention that Halpern discloses "auto-start", [C: 3; L: 42-49; and Fig. 1] there is no disclosure of anything that bears any relation to the recited limitations of claim 1. Halpern does not mention anything that could remotely be interpreted as "auto-start" at the cited location. The Applicant's agent searched the entirety of Halpern and found that there is a mention of an "auto-start utility" [C: 6; L: 49], but this is not related to the recitation of claim 1. Halpern apparently only includes this "auto-start utility" in a list of possible program components and does not describe what it is.

Accordingly, Halpern does not disclose all the limitations of claim 3, and claim 3 is allowable over Halpern.

Claims 4-7

Claims 4 - 7 each recite a type of single action that initiates creation of a self-extracting file.

Since Halpern does not disclose creating a self-extracting file from only a single action, it also does not disclose the types of single actions by a user recited by claims 4 - 7. Claims 4 - 7 are allowable by virtue of their dependence from claim 3 and are also allowable for this additional reason.

Claim 8

Claim 8 is allowable by virtue of its dependence from claim 3.

Claim 9

Claim 9 recites “generating a filename for the self-extracting file, wherein the generated filename is based on a filename associated with the input file.”

Halpern is silent with respect to how the name of the self-extracting file may be based or not be based on the name of an input file.

Claim 9 is allowable by virtue of its dependence from claim 3 and is additionally allowable for reasons similar to those given for claim 2.

Claim 10

Claim 10 is allowable for reasons similar to those given for claim 1. Namely, Halpern does not disclose “automatically creating a self-extracting file configured to automatically launch the received input file responsive to execution of the self-extracting file,” as recited by claim 10.

Claim 20

Claim 20 recites “in response to only a single action, creating a self-extracting file from an input file, wherein the input file is one of a plurality of file types, and automatically selecting a loader based on the input file's type, and wherein the input file will be automatically launched upon execution of the self-extracting file.”

As discussed above with respect to claim 1, Halpern does not disclose an input file being automatically launched upon execution of a self-extracting file.

Moreover, Halpern does not disclose automatically selecting a loader based on the input file's type. The Examiner has asserted that "the limitations of claims 10, 20, 26 and 32 are rejected in the analysis of Claim 1 above, and these claims are rejected on that basis." However, claim 1 does not recite automatically selecting a loader based on an input file's type.

Claim 20 is additionally allowable for reasons similar to those given for claim 1 with respect to “wherein the input file will be automatically launched upon execution of the self-extracting file.”

Halpern does not disclose "automatically selecting a loader based on the input file's type," and moreover does not disclose "wherein the input file will be automatically launched upon execution of the self-extracting file," both of which are recited by claim 20.

Accordingly, Halpern does not disclose all the limitations of claim 20, and claim 20 is allowable over Halpern.

Claim 24

Claim 24 recites a "a third module for creating, in response to only a single action by a user, an executable file from the compressed input file, wherein the input file will be automatically launched upon execution of the executable file."

For reasons similar to those given for claim 1, Halpern does not disclose a "module for creating, in response to only a single action by a user, an executable file from the compressed input file, wherein the input file will be automatically launched upon execution of the executable file."

Halpern only discloses self-extraction, and does not disclose automatic launching of an input file following the self-extraction. Accordingly, Halpern does not disclose all the limitations of claim 24, and claim 24 is allowable over Halpern.

Claim 26

Claim 26 recites "receiving an input file from a user to be used in creating a self-extracting file, wherein the input file is of any file type, and automatically creating a self-extracting file."

In contrast, and as discussed above with respect to claim 1, Halpern does not disclose receiving from a user an input file to be used in creating a self-extracting file. Halpern discloses that a user is allowed to select a subset of program "components and options" available for installation, after which a subset of files corresponding to those components and options are packaged and distributed to the user. Halpern does not

disclose receiving any file from a user. Accordingly, Halpern does not disclose all the limitations of claim 26, and claim 26 is allowable over Halpern.

Claim 27

Claim 27 recites, "receiving, in response to a single action, an input file to be used in creating an executable file, wherein the input file is one of a plurality of file types; and without further instructions, creating an executable file using the received input file, wherein the executable file includes a compressed copy of the input file, and wherein the compressed copy of the input file is automatically decompressed and launched upon execution of the executable file."

For reasons described above with respect to claim 1, Halpern does not disclose "wherein the compressed copy of the input file being automatically decompressed and launched upon execution of an executable file."

Moreover, for reasons similar to those given for claim 1, Halpern also does not disclose receiving an input file in response to a single action.

Accordingly, Halpern does not disclose all the limitations of claim 27, and claim 27 is allowable over Halpern.

Claim 31

Claim 31 recites, "in response to a first action, creating an executable file from an input file, wherein the executable file includes a compressed copy of the input file, and wherein the executable file includes code to decompress and to load the compressed input file; and in response to a second action, executing the executable file to decompress the compressed copy of the input file and launching the decompressed input file with appropriate application software."

In contrast, as discussed above with respect to claim 1, Halpern does not disclose "creating an executable file from an input file, wherein the executable file includes a compressed copy of the input file, and wherein the executable file includes code to decompress and to load the compressed input file." This specific disclosure is

nowhere to be found in Halpern. Moreover, for reasons given for claim 1, Halpern does not generally disclose an "input file is configured to be automatically launched upon execution of the self-extracting file," and therefore also does not disclose the executable file includes "code to decompress and to load the compressed input file," [emphasis added] as recited by claim 31.

Accordingly, Halpern does not disclose all the limitations of claim 31, and claim 31 is allowable over Halpern.

Claim 32

Claim 32 recites "receiving, in response to a single action, an input file to be used in creating a self-extracting file; without further instruction, creating a self-extracting file using the input file and automatically launching the input file upon execution of the self-extracting file."

For reasons described above with respect to claim 1, Halpern does not disclose "receiving, in response to a single action, an input file to be used in creating a self-extracting file; without further instruction, creating a self-extracting file using the input file." Halpern is silent with respect to whether further instruction is required to create a self-extracting file.

Moreover, Halpern does not disclose "automatically launching the input file upon execution of the self-extracting file." Halpern only discloses self-extraction.

Accordingly, Halpern does not disclose all the limitations of claim 32, and claim 32 is allowable over Halpern.

Claim 33

Claim 33 is allowable by virtue of its dependence from claim 32, and for at least the reasons given for claim 32.

Claims 21-23 are rejected under 35 U.S.C § 102(e) as being anticipated by Wygodny et al. [hereafter Wygodny], U.S. Patent No. 6,202,199 based on provisional application No. 60/055,165 filed on July 31, 1997.

Claim 21

Claim 21 recites, "displaying a first frame used to allow a user to specify an input file to be converted to a self-extracting file; receiving the input file specified by the user, wherein the received input file is automatically configured as a self-extracting file, and wherein the input file is automatically launched upon execution of the self-extracting file; and displaying a second frame, wherein the second frame includes a link related to the self-extracting file created from the user specified input file."

Wygodny does not disclose "the input file is automatically launched upon execution of the self extracting file."

Wygodny discloses: "At the end of installation, the user 110 can launch the agent 102." [C: 17; L: 1-2] However, if Wygodny's agent 102 was automatically launched upon execution of the self-extracting file, then the agent would already be launched. Therefore Wygodny, far from disclosing all the limitations of claim 21, teaches away from "the input file is automatically launched upon execution of the self extracting file," as recited by claim 21.

Moreover, Wygodny does not disclose "displaying a first frame used to allow a user to specify an input file to be converted to a self-extracting file," as recited by claim 21.

Wygodny does not disclose "automatically configure[ing] an input file specified by the user as a self-extracting file," as recited by claim 21.

Wygodny does not disclose "displaying a second frame wherein the second frame includes a link related to the self-extracting file created from the user specified input file," as recited by claim 21.

Wygodny discloses a system for tracing the execution paths of a software program without requiring modifications to the executable or source code files of that program. As part of this system, Wygodny teaches providing a small executable

"agent" program that enables a remote user to generate a trace file, and further teaches that the preferred method of providing this "agent" program includes packaging that program as a self-extracting file. These are not the limitations of claim 21.

The Examiner cited Wygodny's FIG. 3A and “col 8, line 51-55; col 17, line 1-7; fig 3A; fig 9-10” to show displaying a first frame used to allow a user to specify an input file to be converted to a self-extracting file. This is incorrect. Referring to FIG. 3A; [C:8; L:50-57], and [C:9; L: 9-13]; Wygodny's frame window 300 (and specifically, executable pane 314) depicts a particular executable file that is currently being traced. It does not allow a user to specify an input file to be converted to a self-extracting file.

The Examiner also cites [C: 17; L: 1-12] to show that Wygodny discloses automatically configuring an input file specified by the user as a self-extracting file. This is also incorrect. Referring to [C: 17; L: 1-12], Wygodny discloses supplying a tracing "agent" program to a user as a self-extracting "zip file.” To install this "agent" program, the user can simply double-click on that zip file. Finally, the user may run the "agent" program, which allows the user to specify both a "Trace Control Information" (TCI) file and a client executable program which the user desires to have traced.

But these are not the limitations of claim 21. Wygodny does not disclose “the received input file is automatically configured as a self-extracting file.” Moreover, Wygodny does not disclose “wherein the input file is automatically launched upon execution of the self-extracting file,” as recited by claim 21.

The Examiner further cites FIGS. 3A-3B and FIG. 5 to show that Wygodny discloses displaying a second frame that includes a link related to a self-extracting file created from a user-specified input file. This is incorrect. Referring to FIGS. 3B and 5, although Wygodny does teach displaying a second frame (relative to the first frame depicted in FIG. 3A) as part of its user interface, none of the user interface frames depicted include a link related to a self-extracting file that has been created from a user-specified input file.

Accordingly, Wygodny does not disclose all the limitations of claim 21, and claim 21 is allowable over Wygodny. “

Claim 22

Claim 22 recites “a receiving module configured to receive an input file, wherein the input file received is one of a plurality of file types and wherein the input file includes an associated filename; a naming module configured to create and name an output file, wherein the output filename is generated from the associated filename of the input file …; a self-extracting module configured to transform the output file into a executable file, wherein the self-extracting module receives the input file and the output file from the naming module; a loader module configured to setup the executable file to launch the input file upon execution of the executable file, wherein the loader module receives the executable file and the input file from the self-extracting module; and a compressing module configured to compress the input file and attach the compressed input file to the executable file, wherein the compressing module receives the input file and the executable file from the loader module.”

Wygodny does not disclose receiving an input file, wherein the input file received is one of a plurality of file types. The Examiner cites “col 9, line 9-13, line 57-62, col 12, line 24-35” to show that Wygodny discloses this limitation. However, to the extent that Wygodny discloses any "receiving module," it does so only with respect to the ability to receive executable files in order to trace the operations of those executable files. Referring, *e.g.*, to [C: 9; L: 9-62] and [C: 12; L: 24-35], the executable files selectable as input files have a single file format .

Wygodny does not disclose “a naming module configured to create and name an output file, wherein the output filename is generated from the associated filename of the input file.” Applicant’s agent has searched Wygodny and finds no mention of an output filename based on an input filename.

Wygodny does not disclose “a self-extracting module configured to transform the output file into a executable file.” Although the Examiner does not provide a citation to show this limitation, Applicant's agent has reviewed Wygodny and can find no instance that discloses transforming an output file into an executable file. Referring to [C: 16; L: 41-44] and [C: 17; L: 1-12], Wygodny discloses distributing the tracing agent program to the user as a self-extracting zip file. That tracing agent program, however, is not "an output file" as recited by claim 22.

Wygodny does not disclose "a loader module configured to setup the executable file to launch the input file upon execution of the executable file."

Wygodny's self extracting file produces a file that is not automatically launched. For reasons similar to those given for claim 21, Wygodny does not disclose a loader module configured to setup the executable file to launch the input file upon execution of the executable file. Wygodny discloses a self-extracting file. The self extracting file produces a file that is not automatically launched. According to Wygodny [C: 17; L: 1-2] "At the end of installation, the user 110 can launch the agent 102." If the agent was automatically launched upon extraction, it would not be necessary for the user to launch the agent.

Accordingly, Wygodny does not disclose all the limitations of claim 22, and claim 22 is allowable over Wygodny.

Claim 23

Claim 23 is allowable at least by virtue of its dependence from claim 22.

Claims 11-19, 25, 28-29, 30, and 34 are rejected under 35 U.S.C. § 103(a) as being unpatentable over Halpern et al. [hereafter Halpern], U.S. Patent No. 6,282,711 filed on Aug. 10, 1999 as applied to claims 10 and 32, above, and further in view of Gage et al. [hereafter Gage], U.S. Patent No. 5,923,846 published on July 13, 1999.

The Examiner has not made a *prima facie* case for obviousness.

Claims 11-19

Claims 11-19 are allowable by virtue of their dependence from claim 10, and for at least the reasons given for claim 10.

Claim 25

Claim 25 recites a tangible medium carrying computer instructions to cause a computer to "provide a compressed input data portion corresponding to an input data file, … and provide a self-extracting stub portion, wherein the self-extracting stub portion is automatically attached to the compressed input data portion and the archive header portion, and wherein the self-extracting stub portion includes a decompression engine to decompress the compressed input data portion, and a loader operable to launch the decompressed input data portion with appropriate application software for handling the input data file."

In contrast, for reasons given with respect to claim 1, Halpern neither discloses nor renders obvious "a loader operable to launch a decompressed input data portion with appropriate application software for handling the input data file," as recited by claim 25.

Gage discloses a system for posting and downloading textual messages and files from an online bulletin board system. Referring, *e.g.*, to FIG. 5 and corresponding text [C: 14; L: 21-67], Gage teaches storing textual messages as compressed RTF (Rich text format) data having a plurality of compressed portions and an uncompressed header portion. However, Gage does not disclose or reasonably suggest using self-extracting files as part of its online bulletin board system.

Since Gage does not disclose a self-extracting file, Gage also does not disclose a self-extracting file that also includes "a loader operable to launch the decompressed input data portion with appropriate application software," as recited by claim 25.

In combination, Halpern and Gage fail to disclose or reasonably suggest "a loader operable to launch the decompressed input data portion with appropriate application software," as recited by claim 25. The limitation is simply nowhere to be found in either reference.

Accordingly, Halpern and Gage, alone and in combination, fail to disclose or reasonably suggest all the limitations of claim 25, and claim 25 is allowable over Halpern and Gage.

Claim 28

Claim 28 recites a process for producing a computer file, comprising: receiving an input file; automatically opening an output file; automatically adding a decompression engine to the output file for decompressing compressed data; automatically adding loader code to the output file for launching the input file with the appropriate application software for handling the input file; … [and] automatically compressing the input file according to a data compression method..."

"Automatically compressing the input file" and "automatically adding a decompression engine to the output file for decompressing compressed data" makes the output file "a self-extracting file," as recited by Claim 1. "Automatically adding loader code to the output file for launching the input file with the appropriate application software for handling the input file" makes the "input file configured to be automatically launched upon execution of the self-extracting file," as recited by claim 1.

For reasons given above for claim 1, Halpern does not disclose or reasonably suggest "creating a self-extracting file using the input file, wherein the input file is configured to be automatically launched upon execution of the self-extracting file." Because of the relationship between the recitations of claim 1 and claim 28 described immediately above, Halpern therefore also does not disclose or reasonably suggest "automatically compressing the input file,…automatically adding a decompression engine to the output file for decompressing compressed data, and … automatically adding loader code to the output file for launching the input file with the appropriate application software for handling the input file," as recited by claim 28.

Gage, for reasons given for claim 25, does not disclose or reasonably suggest a self-extracting file that also includes "a loader operable to launch the decompressed input data portion with appropriate application software." Therefore, Gage also does not disclose or reasonably suggest "automatically adding loader code to the output file for launching the input file with the appropriate application software for handling the input file," as recited by claim 28.

Accordingly, Halpern and Gage, alone and in combination, do not disclose or reasonably suggest all the limitations of claim 28, and claim 28 is allowable over Halpern and Gage.

Claim 29

Claim 29 is allowable by virtue of its dependence from claim 28 and for at least the reasons given for claim 28.

Claim 30

Claim 30 recites "a method for creating an executable file, comprising: in response to a single action, receiving an input file to be used in creating an executable file, wherein the input file is one of a plurality of file types; and without further instruction, creating an executable file using the received input file, wherein the executable file comprises: a compressed input data portion including data compressed according to a data compression method;…and a stub portion, wherein the stub portion is automatically attached to the compressed input data portion and the archive header portion, and wherein the stub portion includes a decompression engine to decompress the compressed input data portion and a loader to launch the decompressed input data portion."

For reasons similar to those given above at least with respect to claim 25, Halpern and Gage, alone and in combination, fail to disclose or reasonably suggest "a method for creating an executable file [including a stub portion], wherein the stub portion includes a decompression engine to decompress the compressed input data portion and a loader to launch the decompressed input data portion."

Accordingly, Halpern and Gage, alone and in combination, fail to disclose or reasonably suggest all the limitations of claim 30, and claim 30 is allowable over Halpern and Gage.

Claim 34

Claim 34 is allowable by virtue of its dependence from claim 32 and for at least the reasons given for claim 32.

For the foregoing reasons, the Appellants request the Board to reverse the Examiner's rejection of claim 25 under 35 U.S.C. § 101; the rejection of claims 1-10, 20, 24, 26, 27, and 31-33 under 35 U.S.C § 102(e); the rejection of claims 21-23 under 35 U.S.C § 102(e); and the rejection of claims 11-19, 25, 28-29, 30, and 34 under 35 U.S.C. § 103(a); and to rule claims 1-34 allowable or remand the application to the Examiner for allowance of claims 1-34.

Dated this 12th day of March, 2010.

Respectfully submitted,

/CAWiklof/

Christopher A. Wiklof
Registration No. 43,990

Customer No. 72455

Graybeal Jackson LLP
400 - 108th Avenue NE, Suite 700
Bellevue, Washington 98004-5565
Telephone: 425.455.5575
Facsimile: 425.455.1046

Enclosures: Appendices A-C

(J) Claims appendix (Appendix A)

Appendix A (the claims appendix) is a copy of all claims involved in the Appeal, as required under 37 C.F.R. § 41.37(c)(1)(viii).

1. (Previously Presented) A method for creating, in response to only a single action by a user enabled electronic device, a self-extracting file, the method comprising:

receiving from the user enabled electronic device, an input file to be used in creating a self-extracting file; and

without further action by the user enabled electronic device, creating a self-extracting file using the input file, wherein the input file is configured to be automatically launched upon execution of the self-extracting file.

2. (Previously Presented) The method of claim 1, wherein the received input file has an associated filename and wherein a filename for the self-extracting file is configured to be automatically generated based in part on the associated filename of the received input file.

3. (Previously Presented) A method for creating, in response to a single action, a self-extracting file from an associated input file, wherein the associated input file is automatically launched upon execution of the self-extracting file, and wherein a user is not required to separately choose a data compression method, create a compressed archive using the chosen compression method, select an input file to be launched upon decompression of the compressed archive, and create a self-extracting file from the compressed archive, the method comprising:

receiving an input file to be used in creating a self-extracting file, wherein the file is one of a plurality of file types; and

in response to only a single action, creating a self-extracting file from the input file, wherein the input file is configured to be automatically launched upon execution of the self-extracting file.

4. (Previously Presented) The method of claim 3, wherein the single action is a single click with a computer pointing device.

5. (Previously Presented) The method of claim 3, wherein the single action is a double-click with a computer pointing device.

6. (Original) The method of claim 3, wherein the single action is speaking a sound.

7. (Original) The method of claim 3, wherein the single action is pressing a key.

8. (Original) The method of claim 3, wherein the single action is a call from a software routine.

9. (Original) The method of claim 3, further comprising generating a filename for the self-extracting file, wherein the generated filename is based on a filename associated with the input file.

10. (Previously Presented) A method for creating a self-extracting file, the method comprising:

receiving a user selection of an input file to be used in creating a self-extracting file, wherein the input file is of any file type; and

automatically creating a self-extracting file configured to automatically launch the received input file responsive to execution of the self-extracting file.

11. (Previously Presented) The method of claim 10, wherein the creation of the self-extracting file comprises:

opening an output file;

attaching a decompression engine to the output file, wherein the decompression engine is capable of decompressing compressed data to a temporary file;

attaching a loader to the output file, wherein the loader configures the output file so as to automatically launch the temporary file after execution of the self-extracting file;

compressing the received input file according to a data compression method;

attaching an archive header including information about the compressed input file; and

closing the output file, wherein the closed output file is the self-extracting file.

12. (Previously Presented) The method of claim 11, wherein the input file is received from a user enabled electronic device.

13. (Original) The method of claim 11, wherein the input file is received from a software routine.

14. (Original) The method of claim 11, wherein the data compression method is the same method for all received input files.

15. (Original) The method of claim 11, wherein the data compression method is determined based on the file type of the received input file.

16. (Previously Presented) The method of claim 11, wherein the loader attached to the output file depends on the file type of the input file.

17. (Original) The method of claim 11, wherein the loader automatically unloads the temporary file.

18. (Original) The method claim 11, further comprising attaching an unloader to the output file to automatically unload the temporary file.

19. (Original) The method of claim 18, wherein the unloader performs cleanup processes on the temporary file.

20. (Previously Presented) A method for creating an executable file, comprising:

in response to only a single action, creating a self-extracting file from an input file, wherein the input file is one of a plurality of file types; and

automatically selecting a loader based on the input file's type; and

wherein the input file will be automatically launched upon execution of the self-extracting file.

21. (Previously Presented) A method of creating a self-extracting file comprising:

displaying a first frame used to allow a user to specify an input file to be converted to a self-extracting file;

receiving the input file specified by the user, wherein the received input file is automatically configured as a self-extracting file, and wherein the input file is automatically launched upon execution of the self-extracting file; and

displaying a second frame, wherein the second frame includes a link related to the self-extracting file created from the user specified input file.

22. (Previously Presented) A system for creating a self-extracting file comprising:

a receiving module configured to receive an input file, wherein the input file received is one of a plurality of file types and wherein the input file includes an associated filename;

a naming module configured to create and name an output file, wherein the output filename is generated from the associated filename of the input file and wherein the naming module receives the input file from the receiving module;

a self-extracting module configured to transform the output file into a executable file, wherein the self-extracting module receives the input file and the output file from the naming module;

a loader module configured to setup the executable file to launch the input file upon execution of the executable file, wherein the loader module receives the executable file and the input file from the self-extracting module; and

a compressing module configured to compress the input file and attach the compressed input file to the executable file, wherein the compressing module receives the input file and the executable file from the loader module;

wherein each module is embodied in hardware, in firmware, or in a collection of software instructions stored in a tangible computer-readable medium.

23. (Original) The system of claim 22, wherein the loader module is further configured to setup the executable file to perform unload processes.

24. (Previously Presented) A system for creating, in response to a single action, a self-extracting file from an associated input file, wherein the associated input file is automatically launched upon execution of the self-extracting file, and wherein a user is not required to separately choose a data compression method, create a compressed archive using the chosen compression method, select an input file to be

launched upon decompression of the compressed archive, and create a self-extracting file from the compressed archive, the system comprising:

a first module for receiving a user selection of an input file to be compressed, wherein the input file is one of a plurality of file types;

a second module for compressing the received input file according to a data compression method; and

a third module for creating, in response to only a single action by a user, an executable file from the compressed input file, wherein the input file will be automatically launched upon execution of the executable file.

25. (Previously Presented) A tangible computer-readable medium carrying computer-executable instructions configured to cause a computer to:

provide a compressed input data portion corresponding to an input data file, the compressed input data portion including data compressed according to a preselected data compression method;

provide an archive header portion, wherein the archive header portion includes information about the compressed input data portion; and

provide a self-extracting stub portion, wherein the self-extracting stub portion is automatically attached to the compressed input data portion and the archive header portion, and wherein the self-extracting stub portion includes

a decompression engine to decompress the compressed input data portion, and

a loader operable to launch the decompressed input data portion with appropriate application software for handling the input data file.

26. (Previously Presented) A method for creating, in response to a single action, a self-extracting file, the method comprising:

receiving an input file from a user to be used in creating a self-extracting file, wherein the input file is of any file type; and

automatically creating a self-extracting file.

27. (Previously Presented) A method for creating an executable file, the method comprising:

receiving, in response to a single action, an input file to be used in creating an executable file, wherein the input file is one of a plurality of file types; and

without further instructions, creating an executable file using the received input file, wherein the executable file includes a compressed copy of the input file, and wherein the compressed copy of the input file is automatically decompressed and launched upon execution of the executable file.

28. (Previously Presented) A process for producing, in response to a single action, a computer file, the process comprising:

receiving an input file;

automatically opening an output file;

automatically adding a decompression engine to the output file for decompressing compressed data;

automatically adding loader code to the output file for launching the input file with the appropriate application software for handling the input file;

automatically adding an archive header to the output file, wherein the archive header includes information relating to the input file;

automatically compressing the input file according to a data compression method;

automatically updating the archive header to include information about the compressed input file; and

automatically closing the output file.

29. (Original) The product produced by the process of claim 28.

30. (Previously Presented) A method for creating an executable file, the method comprising:

in response to a single action, receiving an input file to be used in creating an executable file, wherein the input file is one of a plurality of file types; and

without further instruction, creating an executable file using the received input file, wherein the executable file comprises:

a compressed input data portion including data compressed according to a data compression method;

an archive header portion including information about the compressed input data portion; and

a stub portion, wherein the stub portion is automatically attached to the compressed input data portion and the archive header portion, and wherein the stub portion includes a decompression engine to decompress the compressed input data portion and a loader to launch the decompressed input data portion.

31. (Previously Presented) A method for using an executable file, the method comprising:

in response to a first action, creating an executable file from an input file, wherein the executable file includes a compressed copy of the input file, and wherein the executable file includes code to decompress and to load the compressed input file; and

in response to a second action, executing the executable file to decompress the compressed copy of the input file and launching the decompressed input file with appropriate application software.

32. (Previously Presented) A method for creating a self-extracting file, the method comprising:

receiving, in response to a single action, an input file to be used in creating a self-extracting file;

without further instruction, creating a self-extracting file using the input file and automatically launching the input file upon execution of the self-extracting file.

33. (Original) The method of claim 32, wherein the input file is an executable routine and wherein a function of the executable routine is called upon loading of the executable routine.

34. (Original) The method of claim 32, wherein the input file is a dynamic link library file.

(K) Evidence appendix (Appendix B)

Appendix B includes all evidence submitted pursuant to 37 C.F.R. §§ 1.130, 1.131, or 1.132 or of any other evidence entered by the Examiner and relied upon by Appellants in the Appeal, as required under 37 C.F.R. § 41.37(c)(1)(ix).

No evidence is submitted.

(L) Related proceedings appendix (Appendix C)

Appendix C indicates there are no related interferences, appeals, or judicial proceedings known to Appellant, Appellant's agent, or the Assignee, which are related to, directly affect or are directly affected by, or which have a bearing on the decision of the Board in the pending Appeal.